Learning to Optimize with Graph Neural Networks

Luca Sbicego, Yani Hammache, Nizar Ben Mohamed
School of Computer Science and Communication Systems, EPFL, Switzerland

***Abstract*—We investigate the advantages and limitations of graph neural networks (GNN) as MetaOptimizers in the learning-to-optimize setting (L2O). Extending existing experiments, we benchmark GNN-based optimizers against L2O and traditional baselines and test out architectural refinements. Notably, we also study the generalization behavior of different approaches by optimizing neural networks of increasing complexity. Our findings indicate that GNNs perform on par with baselines when the tasks is similar, but exhibit massive performance advantages when optimizing larger networks. These findings highlight the potential but also the limitations of GNNs in L2O, motivating further research.**

I. Introduction

Learning to optimize (L2O) reframes the design of optimizers as a data-driven problem. Instead of hand-crafted parameter rules (as in Adam, RMSProp, etc.), a *MetaOptimizer* —typically parametrized by a neural network— is trained to produce parameter updates that minimize a given loss on a target model, called the *optimizee* [1]. MetaOptimizers often outperform classical methods on benchmarks. Under the hood, they learn to perform momentum-like accelerations, learning rate scheduling as well as gradient preconditioning or clipping, more effectively adapting to the optimization landscape than fixed optimization algorithms [2].

Early L2O approaches built on recurrent neural networks, with [3] first proposing an LSTM architecture that updates each optimizee parameter independently based on its past gradients. Extensions include hierarchical architectures, aiming to capture interactions between optimizee layers by passing summary statistics to higher-level recurrent networks. More recent work also includes transformer-based solutions [4].

Such existing L2O approaches typically act on individual parameters, ensuring permutation-invariance towards the optimizee's neurons and applicability of the trained optimizers to different optimizee architectures. [5] suggested to interpret neural networks as graphs, with nodes representing neurons and edges the connections. They propose to leverage Graph Neural Networks (GNNs) to learn powerful equivariant representations of neural networks, with obvious application to L2O. Such MetaOptimizers naturally account for the network structure and interactions between parameters and layers, while still being computationally efficient. Their approach is briefly outlined in the following section.

While promising, their work includes limited L2O-specific experiments [5]. Hence, we build on this GNN framework for meta-learning and extend the original experiments in three key ways: **(1) Reproduction:** we rigorously reproduce previous research and provide comparisons against established baselines; **(2) Improving the Approach:** we propose architectural and training regime refinements that could potentially enhance stability and convergence speed; and **(3) Generalization to Diverse Tasks:** We evaluate generalization aspects by optimizing network architectures of progressively greater complexity and show that equivariant GNN-based optimizers perform more reliably than competing methods.

II. Neural Graphs for L2O

This sections provides an overview of the method proposed in [5]. Additional details are reported in Appendix A.

Graph Construction: For a network with L layers, the node features are set as the biases $b^{(l)} \in \mathbb{R}^{d_l}$, and the edge features as the weights $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$. These are combined into global feature matrices: $V \in \mathbb{R}^{n \times d_V}$ and $E \in \mathbb{R}^{n \times n \times d_E}$, where $n = \sum_{l=0}^{L} d_l$ is the total number of neurons across all layers. The adjacency structure reflects the network's feedforward connectivity, and the weight matrices populate the block-sparse tensor E in a layer-wise manner. This graph structure ensures that functionally equivalent networks—those differing only by hidden neuron permutations—map to isomorphic graphs, thereby enabling permutation-equivariant processing.

Features Construction: Features are expanded with dynamic per-parameter features such as gradients $\nabla W^{(l)}$ and multi-scale momentums $\beta \in \{0.5, 0.9, 0.99, 0.999, 0.9999\}$, which are in turn processed using log and sign functions. To capture functional behavior in addition to structural properties, the authors additionally introduce *probe features*. These are derived by feeding a set of learned input vectors $x \in \mathbb{R}^{d_0}$ into the neural network and recording all intermediate activations. We defer to the appendix or original paper for details.

MetaOptimization: The output of the model is a per-parameter update $\Delta W_{ij} = \text{GNN}(e_{ij}, v_i, v_j)$, predicted for each edge and node in the graph based on the features of the edge e_{ij} and the features of the sender and receiver nodes v_i and v_j. By learning directly on the parameter graph of the optimizee network, this method enables architectural generalization and inductive bias toward symmetry-preserving solutions.

Models: Both GNNs and Transformers are naturally suited to operate over graph-structured data, as a transformer can be seen as a GNN operating over the fully-connected input graph. As in the original work, we consider the PNA architecture [6] with additional edge feature updates as the GNN, and the relational transformer architecture [7], which considers edge features for self-attention, as the transformer.

III. Experimental Setup

Training: All models are trained with $1,000$ outer steps on the FashionMNIST dataset [8]. At each outer step, an "opti-

mizee" CNN is trained for 100 inner steps, if not otherwise specified. Those gradients wrt to the task loss are unrolled and backpropagated through the MetaOptimizers as learning signal. MetaOptimizers themselves are trained by Adam with an initial learning rate of $3e-4$ and a Cosine scheduler.

Evaluation: Given a that MetaOptimizers are notoriously hard to train, we consider all model checkpoints every 100 outer steps and pick the best performing one based on the train task (i.e., we pick the best performing model FashionMNIST validation set). Our evaluation protocol consists of optimizing a slightly larger CNN (64 hidden dimensions vs 32 used in training) on CIFAR10 [9] for $1,000$ epochs over 5 different seeds.

We propose three different experiments to further establish the neural graph-based approach [5] to the L2O field:

1. Replication and Comparisons against Appropriate Baselines: In a first step, we reproduce results from the original paper, training and evaluating the `GNN`, Relational Transformer `RT` operating on the fully connected graph, as well as the L2O baselines `LSTM` and `MLP`. The latter two treat each parameter update independently, without considering the network structure. Additionally, we include comparisons against properly tuned Adam [10] and RMSProp optimizers. We perform a grid search over learning rates $[0.05, 0.002, 0.001, 0.005, 0.001]$ and weight-decay $[0, 1e-3, 1e-5]$ and pick the best performing hyperparameters on the validation set.

2. Improvements to Proposed Approach: We propose two new model types and training regimes. Firstly, we design a composite `RecurrentGNN` architecture, where we apply the GNN on the outputs of an LSTM cell. This approach could potentially combine advantages of learning temporal patterns together with the topological bias of the GNN. Secondly, training MetaOptimizers on a fixed number of inner steps inevitably leads to bias, as the window in which gradients are unrolled is fixed [11]. To counteract this, we train a `GNN-variable` model where we randomly sample the number of inner training steps between 75 and 250.

3. Investigation of Generalization Advantages of GNN-based MetaOptimizers: Classical MetaOptimizers only exhibit limited transferability to different tasks and optimizee' architectures, as they treat each parameter update independently without considering the network structure. Therefore, graph-based MetaOptimizers might generalize better. To study this conjecture, we optimize CNNs of varying depth and width on CIFAR10 with our trained MetaOptimizers. We plan to analyze performance differences between `GNN`, `LSTM` and `RT` as the optimizee model complexity changes.

IV. Results

A. Replication & Baselines

Figure 1 presents the convergence plots (accuracy vs. training steps) of the optimizers introduced in the previous section. On the CIFAR10 test tasks, the simpler `LSTM` performs on par with the `GNN` leveraging the neural graphs approach. Both methods clearly outperform all others by a large margin, with the `RT` being the runner up. The simpler `MLP` L2O-baseline clearly underperforms even the handcrafted optimizers (Adam and RMSProp).

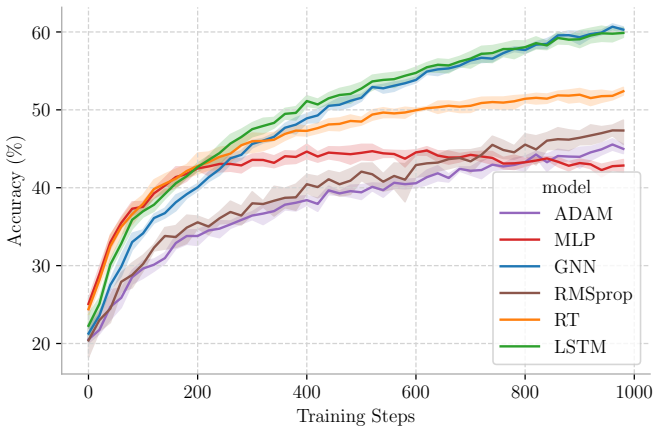

Fig. 1. Accuracy (avg $\pm$ std) over optimization process on CIFAR10 for a 3-layer CNN with 64 hidden channels achieved by different optimizers.

As such, we do not fully replicate the results reported in [5] (cf. Figure 3 in the original paper), where the GNN and RT both outperform the LSTM baseline, despite using the same hyperparameters as reported by the authors. Likely, randomness in the MetaOpt training process has a large influence on final performance.

B. Proposed Improvements

Figure 2 shows the training curves on CIFAR10 for the new proposed model types `RecurrentGNN` and `GNN-variable` compared the original `GNN` model (in blue).

Evidently, training the GNN with a randomized number of inner steps did not translate in any benefits in our standard evaluation setup, performing consistently below its original variant. Interestingly, the `RecurrentGNN` exhibits faster progress in the initial iterations, before diverging around step 100 — precisely the number of inner steps used during training. We conclude the approach shows some promise in principle, yet, the increased complexity (LSTM combined with GNN) seems to result in some generalization instabilities. This could potentially be addressed by further optimizations (e.g., hyperparemeter search, multi-task training regimes), which is out of the scope of this project due to computational constraints.

C. Generalization Behviour

Figure 3 shows the results of our generalization study for varying hidden dimensions (top) and network depth (bottom). We plot the advantage of the `GNN`, `RT` and `LSTM` over the deterministic Adam-baseline, defined as $y = \textit{model\_performance} - \textit{adam\_performance}$ for each testing setup.

Both the `GNN` as a graph-based method `LSTM` as a simpler method reach their biggest advantage over Adam when optimizing similarly-sized networks than encountered during

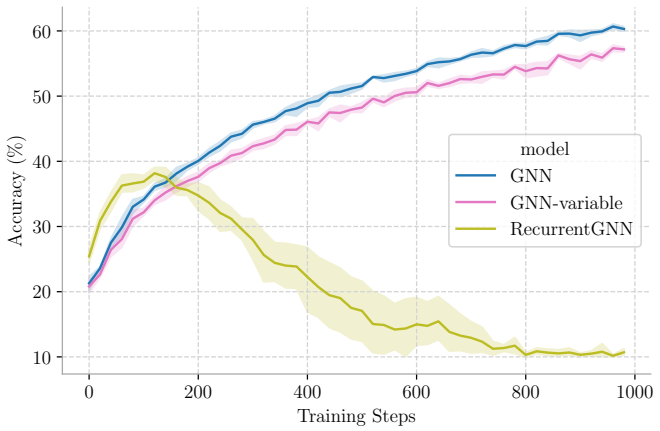

Fig. 2. Accuracy (avg ± std) over optimization process on CIFAR10 for a 3-layer CNN with 64 hidden channels achieved by proposed improved approaches.

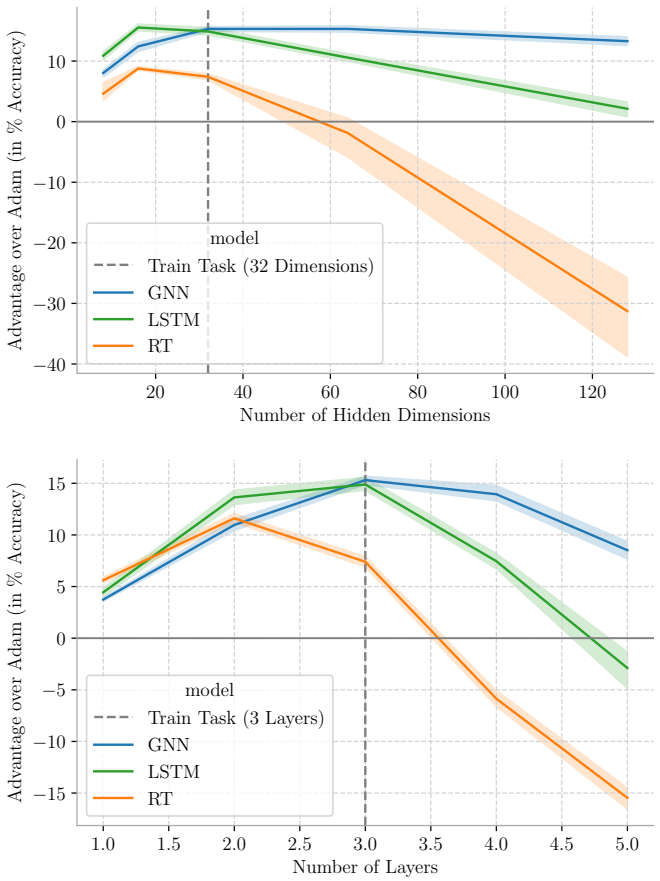

Fig. 3. Performance advantage over Adam of L2O models (avg ± std) for varying model complexities.

training (32 hidden dimensions and 3 layers as indicated by the vertical dashed line). However, as soon as optimizee complexity increases (increased hidden dimensions or depth) the GNN outperforms drastically, indicating that it successfully leverages the interdependencies between the parameters as captured by the graph topology. The LSTM — treating each parameter independently — starts to fall even behind Adam when optimizing increasingly larger networks. While GNN-based methods might not always outperform the LSTM, this establishes their substantive advantage when it comes to generalizing models of increased complexities.

The relational transformer RT on the other hand operates on the fully connected graph and thus has a $\mathcal{O}(n^2)$ dependency in the number of neurons n. Likely, it "overwhelmed" by these rapidly increasing dependencies, explaining its divergence when optimizing models of increased complexity (esp. for larger hidden dimensions).

V. Discussion & Conclusion

In this paper, we replicated and significantly expanded initial experiments around leveraging neural graphs to learn equivariant representation of neural networks in the L2O field [5]. After replicating the original setup, we tested out two architectural improvements to the originally proposed approach and studied the generalization behavior of the trained MetaOptimizers versus simpler baselines. We derive three main implications from our experiments.

1. Neural graph-based approaches are not always performing better out of the box. In our experiments, a simpler LSTM baseline performs on par with more complex GNN-based methods, contrary to results reported in [5]. This highlights the need for careful evaluation protocols, especially as L2O models are notoriously tricky to train.

2. GNN-based MetaOptimizers show massive generalization advantages over traditional L2O approaches. Leveraging the topology of neural networks, such models manage to train optimizees of increased complexity compared to what they encountered during training. Simpler approaches fail at this task. Studying *why* GNN-based optimizers generalize better — by reverse engineering learned patterns analog to [2] — could yield interesting insights, which we leave for future work.

3. Approaches modeling temporal as well as topological features could be promising avenues for future research. Our proposed RecurrentGNN (LSTM combined with GNN) reaches superior performance on the initial trajectory but consistently diverges in later iterations of the optimization process. More careful training might alleviate such issues, potentially yielding a novel L2O architecture.

Our findings are naturally limited by severe compute constraints that have only allowed for restricted hyperparameter exploration. Furthermore, our analysis constrains itself to tasks in the computer vision field, neglecting questions of generalization across optimizee model families (e.g., CNNs vs Transformers) or multi-task training. Nonetheless, given the limited depth of experiments in the original paper, this study has significantly extended the understanding of the limitations potential for applying geometric deep learning approaches to the fast moving L2O field. Further research needs to be conducted to explore generalization behavior more broadly and establish robust and high-performing equivariant MetaOptimizers.

References

[1] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, 2022.

[2] N. Maheswaranathan, D. Sussillo, L. Metz, R. Sun, and J. Sohl-Dickstein, "Reverse engineering learned optimizers reveals known and novel mechanisms," 2021. [Online]. Available: https://arxiv.org/abs/2011.02159

[3] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/fb87582825f9d28a8d42c5e5e5e8b23d-Paper.pdf

[4] A. Moudgil, B. Knyazev, G. Lajoie, and E. Belilovsky, "Learning to optimize with recurrent hierarchical transformers," in *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*, 2023.

[5] M. Kofinas, B. Knyazev, Y. Zhang, Y. Chen, G. J. Burghouts, E. Gavves, C. G. M. Snoek, and D. W. Zhang, "Graph neural networks for learning equivariant representations of neural networks," 2024. [Online]. Available: https://arxiv.org/abs/2403.12143

[6] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *Advances in neural information processing systems*, vol. 33, pp. 13 260–13 271, 2020.

[7] C. Diao and R. Loynd, "Relational attention: Generalizing transformers for graph-structured tasks," *arXiv preprint arXiv:2210.05062*, 2022.

[8] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[9] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[10] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] L. Metz, N. Maheswaranathan, J. Nixon, C. D. Freeman, and J. Sohl-Dickstein, "Understanding and correcting pathologies in the training of learned optimizers," 2019. [Online]. Available: https://arxiv.org/abs/1810.10180

Appendix

Details Neural Graphs for L2O

Equivariance: An *equivariant representation* means that the representation of a neural network remains consistent under permutations of its hidden neurons. Formally, for a group action π on the neurons, the representation f satisfies:

$$f(\pi \cdot \theta) = \pi \cdot f(\theta),$$

where θ denotes the parameters of the input neural network.

CNNs as Graphs: For convolutional neural networks (CNNs), filters are first zero-padded to a maximum size $(w_{\text{max}}, h_{\text{max}})$ and then flattened so they can be uniformly treated as edge features. This allows varying kernel sizes to coexist in a unified graph representation. Additional architecture-specific information, such as activation functions or residual connections, is included using categorical embeddings or added edges.

L2O Feature Construction The graph features include dynamic per-parameter features such as gradients $\nabla W^{(l)}$ and multi-scale momentums:

$$\beta \in \{0.5, 0.9, 0.99, 0.999, 0.9999\},$$

which are preprocessed using log and sign functions. Thus, the edge feature vector may include:

$$e_{ij} = \left[W_{ij}, \nabla W_{ij}, \text{momentum}_{ij}^{(1)}, \ldots, \text{momentum}_{ij}^{(5)}\right].$$

To capture functional behavior in addition to structural properties, the authors introduce *probe features*. These are derived by feeding a set of learned input vectors $x \in \mathbb{R}^{d_0}$ into the neural network and recording all intermediate activations. For example, in a 2-layer MLP:

$$f(x) = W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)},$$

the probe feature vector is:

$$V^{\text{probe}} = \left(x, \sigma(W^{(1)}x + b^{(1)}), f(x)\right)^T.$$

These vectors are appended to each node's features, enriching the representation with task-relevant behavior.